

Epidemic Information Diffusion on Complex Networks

Laura Ricci

Dipartimento di Informatica

25 luglio 2012

PhD in Computer Science

Outline

- 1 Introduction
- 2 Epidemic virus diffusion: models
- 3 Epidemic algorithms
- 4 Gossip algorithms

Epidemic Information Diffusion

Epidemic (or gossip computational) paradigm

- stochastic approach oriented towards large scale system characterized from a high dynamicity
- main characteristics:
 - scalability
 - simplicity
 - efficiency
 - robustness: resilience with respect to node faults/loss of messages/ network partitioning
 - probabilistic behaviour

Epidemic Protocols

- first proposal: an epidemic protocol to update a replicated data base proposed by Alan Demers et al. in 1987 at Xerox
- in the last decade this approach has been applied to P2P networks
- currently, different implementations in:
 - *Amazon S3 (Simple Storage System)*: gossip is exploited to detect available servers and node state.
 - *Amazon Dynamo*: failure detection e membership service
 - *Cassandra*: distributed database exploited by some early version of Facebook
 - *Bittorrent*: information diffusion within a swarm

Epidemic Protocols: Characteristics

Top level behaviour of an epidemic protocol:

- each node periodically contacts one (or more) node chosen uniformly at random and exchange information with them
- the nodes to be contacted may be chosen
 - among the neighbours on the network graph
 - through a random peer sampling service implemented by a random walk or by a gossip protocol

Diffusion of the information on the network

- bio-inspired: mimics the diffusion of a virus in the network
- self-stabilizing: no centralization point
- self-healing: explicit mechanisms for the management of dynamicity (churn) are required

Epidemic Approaches

- *Anti Entropy*
 - epidemic diffusion of the state updates of the nodes
 - nodes periodically choose another node and update their state. Used to make the state of a distributed data base consistent
- *Rumour Mongering*
 - mongering= spreading the rumour
 - nodes are "infected" by a rumour and propagate it to other nodes
 - first model for the diffusion of a virus in a population

Epidemic Virus Diffusion: SIR Model

The system evolves according to a set of simple rules:

- an infected individual coming into contact with a healthy individual may infect him/her with a given probability

SIR Model:

- *Susceptible*: has not caught the virus, but may become infected put in contact with an infected individual
- *Infective*: has caught the virus and can transmit it
- *Recovered*: an individual which has recovered and acquired a permanent immunity



Epidemic Diffusion: Other Models

- *Susceptible-Infective-Removed-Susceptible (SIRS)* :
temporary immunity: an individual may be
infected, recover and then catch again the virus



- *Susceptible-Infective (SI):* State Exposed: models incubation



- *Susceptible-Infective (SI):* When an individual becomes infected it remains in this state until all population is infected



Modelling Rumour Mongering

A model for epidemic diffusion:

- initially n individuals, everyone is susceptible
- one of them gets acquainted of a news (rumour) and become infected
- contacts another individual and 'share the rumour'
- each infected individual shares the rumour, in turn
- if an individual contacts a given number of other individual which are already infected, it "looses its interest" in spreading the rumour and becomes removed
- when does the system converge? How may cycles are needed in order to have no infected individual? And in this state how may individual know the rumour?

Modelling Rumour Mongering

Let

- S_t number of susceptible individual at time t
- I_t number of infected individual at time t
- R_t number of removed individual at time t
- N population size
- $s_t = \frac{S_t}{N}$ fraction of the susceptible individuals on the entire population
- $i_t = \frac{I_t}{N}$ fraction of infected individuals
- $r_t = \frac{R_t}{N}$ fraction of removed individual

The following relations hold:

$$S_t + I_t + R_t = N$$

$$s_t + i_t + r_t = 1$$

Modelling Rumour Mongering

- System parameters
 - γ average number of individuals contacted by an infected individual
 - α percentage of susceptible individuals that, if contacted by an infected individual, become infected
 - $\beta = \gamma \times \alpha$, trasmission parameter
 - percentage of individuals infected by an infected one
 - κ probability that an infected individual becomes immune
- the contacts between individuals belonging to the three groups are uniformly distributed at random
- if an infected individual contact another infected individual or a removed individual, no changes in the systems
- if an infected individual contacts a susceptible individual, a new contagion occurs

Modelling Rumour Mongering

- The evolution of the system in time is modelled by a system of *differential equations* (derivatives computed with respect to time)
- If we consider a discrete sequence of time, we obtain

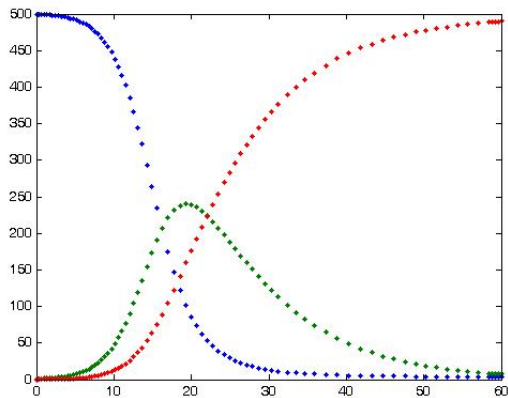
$$\begin{cases} S_{t+1} = S_t - \beta \times s_t \times I_t \\ I_{t+1} = I_t + \beta \times s_t \times I_t - \kappa I_t = I_t(1 + \beta s_t - \kappa) \\ R_{t+1} = R_t + \kappa I_t \end{cases}$$

- The second equation considers the infected individual which become immune

Epidemic Virus Diffusion: SIR Model

The system may be described by a set of *differential equations*

- initially, a single infected individual
- the susceptible becomes infected
- the infected individual become removed



A Simple Epidemic Algorithm

- Given a network $G = (V, E)$ of n nodes, an arbitrary node $v \in V$ has a piece of information that it wishes to spread to all the other nodes as quickly as possible
- Let $P = P_{ij}$ be the transition probability matrix paired with the graph.
- Algorithm: Let time be discrete and denoted by $t \in N$
 - Let $S(t) \subset V$ denote the set of nodes that contain node v 's information at time t . Initially, $t = 0$ and $S(0) = \{v\}$.
 - at time $t \geq 1$ each node $i \in S(t)$ contacts one of its neighbours, say j with probability P_{ij} ; it will not contact other any other node with probability P_{ii} .
 - *Push Pull behaviour*: upon contacting, if either i or j had v 's information at time $t - 1$, then both will have v 's information at the end of time t

A Simple Epidemic Algorithm

The algorithm is very simple, but it opens ... some interesting issues:

- a rule for stopping the information diffusion should be defined
 - heuristics: when a node has contacted a predefined number of already infected nodes it loses its interest in propagating the information
- evaluation of the coverage speed: how many steps are required before all the nodes have received the information?
 - probabilistic results depending on the probability matrix
 - the algorithm can be modelled as a set of parallel random walk: requires more complex technique with respect to the simple complete graph covering by a single random walk

Epidemic Algorithm: Graph Covering

We are considering a stochastic process: in general we will obtain probabilistic bounds.

- For each $v \in V$, we consider the case where the epidemic diffusion starts from v .
- For $\epsilon > 0$, we wish to find the time when all nodes have node v 's information with probability at least $1 - \epsilon$
- T_{spr} , coverage time of the graph, which is a function of ϵ , may be defined as follows:

$$T_{spr}(\epsilon) = \sup_{v \in V} \inf \{t : \mathbb{P}(S(t) = V \mid S(0) = \{v\}) > 1 - \epsilon\}$$

Epidemic Algorithms: Conductance

- Let us consider a random walk on a set of states S and stationary distribution π , for each edge consider the *ergotic flow*:

$$q(x, y) = \pi(x)P(x, y)$$

- Ergotic flow from a set $A \subset S$ and A^c :

$$\sum_{x \in A, y \in A^c} q(x, y) = \sum_{x \in A, y \in A^c} \pi(x)P(x, y)$$

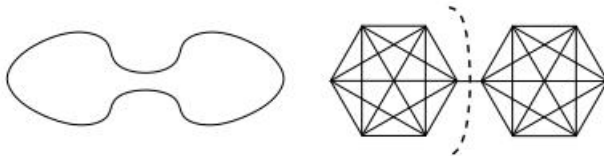
- The *conductance* $\phi(P)$ of the random walk is defined as:

$$\phi(P) = \min_{A \subset S} \frac{\sum_{x \in A, y \in A^c} \pi(x)P(x, y)}{\pi(A) \times \pi(A^c)}$$

where $\pi(A) = \sum_{i \in A} \pi(i)$.

Epidemic Algorithms: Graph Covering

- intuition tells us that if there are a lot of edges and no bottlenecks then the flow should spread out quickly



- a small conductance corresponds to a bottleneck in the graph which obstacles the ergotic flow
- a graph with a low conductance will have a large coverage time

A Coverage Bound for a Symmetric Matrix

- the bounds on T_{spr} depends on the structure of the graph and of the corresponding probability matrix
- for instance the following theorem holds for symmetric matrix (the probability of going from state i to state j is the same of going from state j to state i)

Theorem

Let P is a stochastic aperiodic, irreducible and symmetric matrix defined on a graph G . Then $T_{spr} = O(\frac{\log n + \log \epsilon^{-1}}{\Phi(P)})$, where $\Phi(p)$ is the conductance of the probability matrix P

A Random Peer Sampling Service

- Epidemic algorithms require a peer is able to contact another peer chosen at random
- this implies a random peer sampling service which may be implemented
 - by a Random Walk: the walker visit the graph and reports a sample of the nodes
 - by a simple *gossip algorithm*

A Random Peer Sampling Service

- each node has a partial view of the network including c neighbours
- each node periodically contacts a neighbour in its view and exchange with it some informations
- the exchanged information regard the view of the nodes
- the nodes gossip with their neighbours and the subject of gossip is ... their neighbours!
- view shuffling: the view of a node changes dinamically
- natural self-healing of the network

Newcast: a Simple Random Peer Sampling Protocol

Active thread

```
selectPeer(&Q);  
selectToSend(&bufs);  
sendTo(Q, bufs);  
  
receiveFrom(Q, &bufr);  
selectToKeep(p_view, bufr);
```

Passive thread

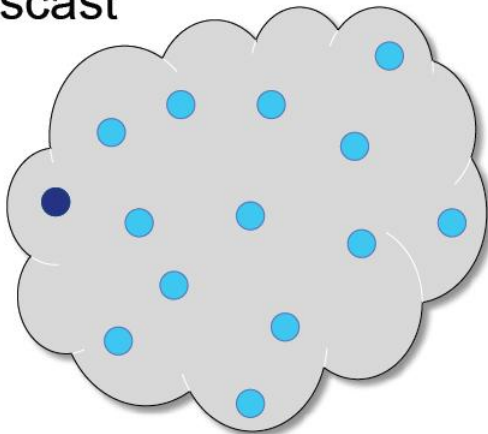
```
receiveFromAny(&P, &bufr);  
selectToSend(&bufs);  
sendTo(P, bufs);  
selectToKeep(p_view, bufr);
```

- SelectPeer: selects at random a peer from the local view
- SelectToSend: selects some entries from the local view
- SelectToKeep: merges of the received information with the local view, duplication elimination, selection of a subset of the resulting view

Note: A timestamp is increased at each gossip cycle: fresher descriptors have higher timestamps!



Newcast: a Simple Random Peer Sampling Protocol

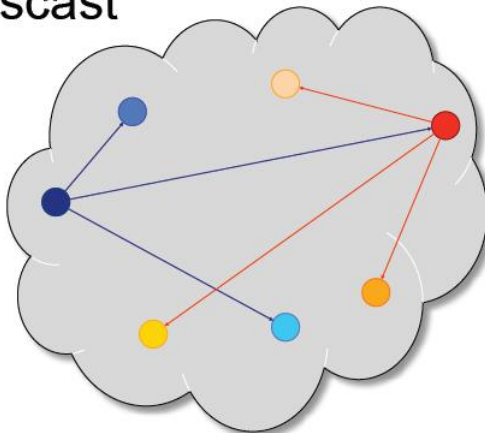
Newscast



Newcast: a Simple Random Peer Sampling Protocol

Newscast




ID & Address	Time stamp
	9
	12
	16

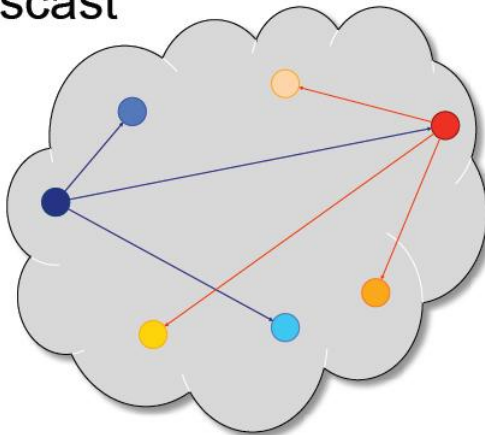


ID & Address	Time stamp
	7
	10
	14

Newcast: a Simple Random Peer Sampling Protocol

Newscast

ID & Address	Time stamp
	9
	12
	16






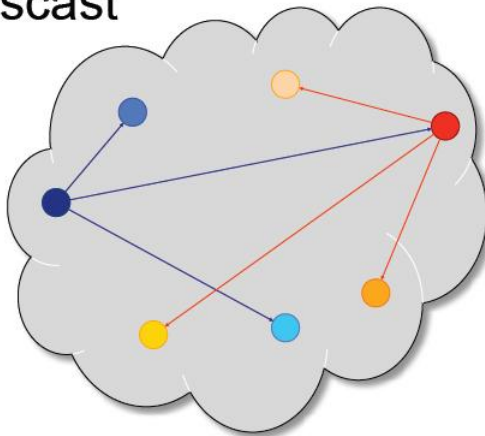
ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view

Newcast: a Simple Random Peer Sampling Protocol

Newscast

ID & Address	Time stamp
	9
	12
	16






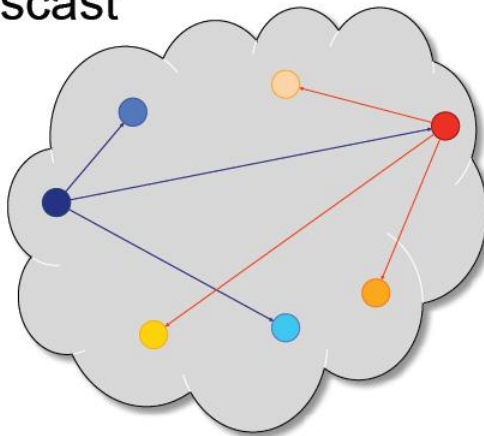
ID & Address	Time stamp
	7
	10
	14

1. Pick random peer from my view

Newcast: a Simple Random Peer Sampling Protocol

Newscast

ID & Address	Time stamp
	9
	12
	16






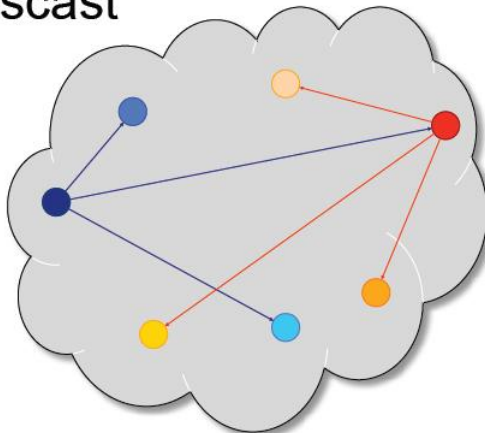
ID & Address	Time stamp
	7
	10
	14


1. Pick random peer from my view
2. Send each other view + own fresh link


Newcast: a Simple Random Peer Sampling Protocol

Newscast

ID & Address	Time stamp
	9
	12
	16



ID & Address	Time stamp
	7
	10
	14




	9
	12
	16




	20
---	----


1. Pick random peer from my view
2. Send each other view + own fresh link

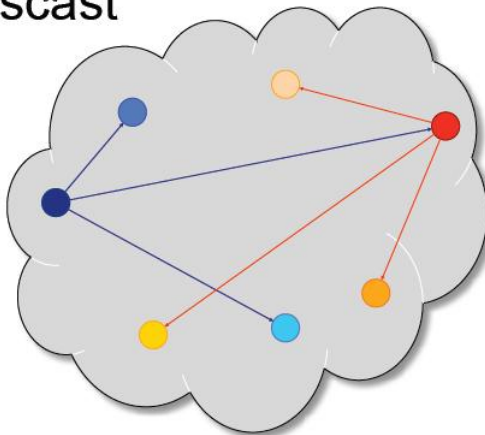
Newcast: a Simple Random Peer Sampling Protocol

Newscast

ID & Address	Time stamp
	9
	12
	16

	7
	10
	14

	20
---	----



ID & Address	Time stamp
	7
	10
	14




	9
	12
	16




	20
---	----


1. Pick random peer from my view
2. Send each other view + own fresh link

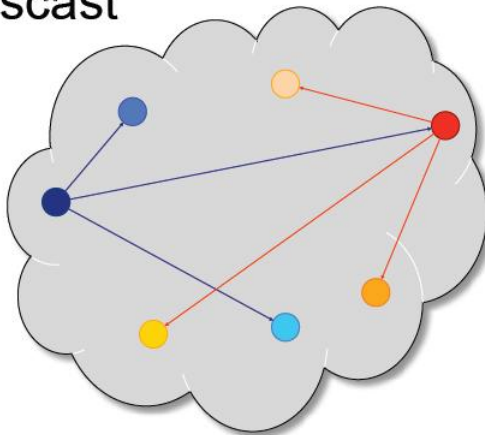
Newcast: a Simple Random Peer Sampling Protocol

Newcast

ID & Address	Time stamp
	9
	12
	16

	7
	10
	14

	20
---	----



ID & Address	Time stamp
	7
	10
	14








	9
	12
	16

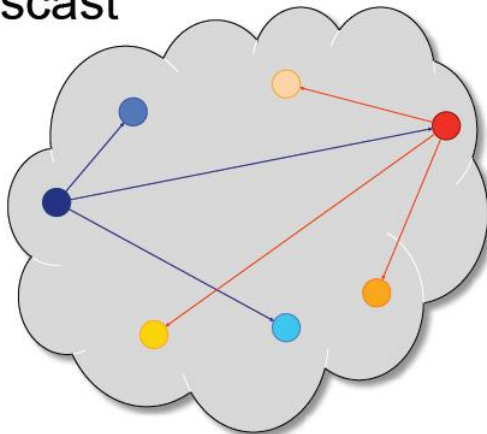
	20
---	----

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest links (remove own info, duplicates)

Newcast: a Simple Random Peer Sampling Protocol

Newscast

ID & Address	Time stamp
	9
	12
	16
	7
	10
	14
	20






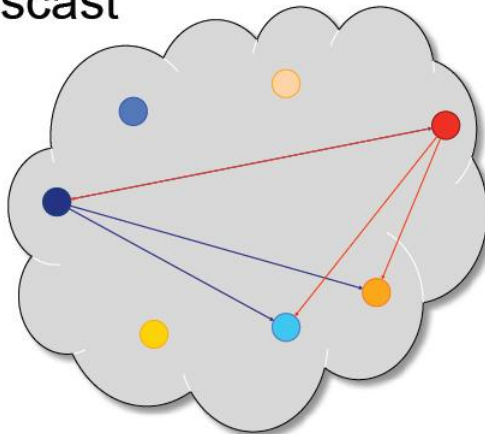
ID & Address	Time stamp
	7
	10
	14
	9
	12
	16
	20

1. Pick random peer from my view
2. Send each other view + own fresh link

Newcast: a Simple Random Peer Sampling Protocol

Newscast

ID & Address	Time stamp
	14
	16
	20



ID & Address	Time stamp
	14
	16
	20

1. Pick random peer from my view
2. Send each other view + own fresh link
3. Keep c freshest link (remove own info, duplicates)

A More Refined Peer Sampling Service

- A more refined Peer Sampling Service has been proposed in "Gossip-based Peer Sampling, ACM Transactions on Computers", M. Jelasity, S. Voulgaris, R. Guerraoui, A.M. Kermarrec, M. Van Steen,
- a protocol similar to Newscast, but more flexible
- goals of the protocol
 - efficient computation of a dynamic changing partial view of the network
 - efficient management of network churn
 - protocol tuning by setting a set of parameters
 - push/pull/pushpull strategy selection
- Note: The timestamp of each of descriptor is increased at each gossip cycle, fresher descriptors have smaller timestamps!

A More refined Peer Sampling Service

- the local view of a node includes a set of node descriptors
 - node descriptor = $\langle IP\ Address, descriptor\ age \rangle$
- protocol Parameters
 - n , number of nodes of the network
 - c local view size
- protocol operations

SelectPeer()

return a peer in the view

Permute()

random shuffle of the view

IncreaseAge()

increase the age of each descriptor (+1)

Append()

add a set of descriptors at the end of the view

RemoveDuplicates()

remove duplicates and maintains the fresher descriptors

RemoveOldItems(n)

remove the n oldest descriptors

RemoveHead(n)

remove the first n descriptors

RemoveRandom(n)

removes n descriptors at random

The Active Thread

```
do forever
  wait(T time units) // T is called the cycle length
   $p \leftarrow \text{view.selectPeer}()$  // Sample a live peer from the current view
  if push then // Take initiative in exchanging partial views
    buffer  $\leftarrow (\langle \text{MyAddress}, 0 \rangle)$  // Construct a temporary list
    view.permute() // Shuffle the items in the view
    move oldest H items to end of view // Necessary to get rid of dead links
    buffer.append(view.head(c/2)) // Copy first half of all items to temp. list
    send buffer to  $p$ 
  else // empty view to trigger response
    send (null) to  $p$ 
  if pull then // Pick up the response from your peer
    receive buffer $_p$  from  $p$ 
    view.select(c,H,S,buffer $_p$ ) // Core of framework – to be explained
  view.increaseAge()
```

The Active Thread

- view: the local view
- buffer: contains the descriptors to send
- each peer
 - add its own descriptor with the freshest time-stamp to the buffer to send, then shuffle the view
 - put H oldest descriptors to the end of the view.
 - In this way, the probability to send "old descriptors" is low.
 - It is possible to tune the "oldness" of a descriptor by a proper tuning of H .
 - send the first $\frac{c}{2}$ descriptors to the partner
- the core of the protocol is the *select* procedure

Design Choices

Peer selection choices:

- select a peer uniformly at random from the view
- select the node with the highest age. It is the peer which has been contacted less recently. Dynamic overlays.
- select the peer which has been contacted more recently. Low probability to find new information in the received view and static overlays.

Design Choices

View propagation choice:

- *push*: the node sends a set of descriptors to the selected peers
- *pushpull*: the node and the selected peer exchange descriptors
- *pull*: a node ask another node for the descriptors.
 - a node cannot inject information about itself, except only when explicitly asked by another node
 - when a node loses all its incoming connections there is no possibility to reconnect to the network
 - for this reason this strategy is seldom exploited

The Passive Thread

```
do forever
  receive bufferp from p // Wait for any initiated exchange
  if pull then // Executed if you're supposed to react to initiatives
    buffer ← (⟨ MyAddress, 0 ⟩) // Construct a temporary list
    view.permute() // Shuffle the items in the view
    move oldest H items to end of view // Necessary to get rid of dead links
    buffer.append(view.head(c/2)) // Copy first half of all items to temp. list
    send buffer to p
  view.select(c, H, S, bufferp) // Core of framework – to be explained
  view.increaseAge()
```

View selection

```

method view.select( c, H, S, bufferp )
    view.append(bufferp) // expand the current view
    view.removeDuplicates() // Remove by duplicate address, keeping youngest
    view.removeOldItems( min(H,view.size-c) ) // Drop oldest, but keep c items
    view.removeHead( min(S,view.size-c) ) // Drop the ones you sent to peer
    view.removeAtRandom(view.size-c) // Keep c items (if still necessary)

```

- returns the new local view of the peer
- parameters:

c	local view size
$H(\text{Healing})$	number of "old" descriptor moved at the end of the view
$S(\text{Swapped})$	number of elements deleted at the head of the list.
	Controls how many of the elements exchanged
	with the partner are deleted from the view
$buffer_p$	received buffer

View Selection

- view Selection: critical parameters, H , S
- suppose c is even. After having appended to the local view the view received from the neighbour, the size of the local view is $c + \frac{c}{2}$
- to reduce the resulting view to the required size of $\frac{c}{2}$
 - delete duplicates
 - delete the oldest H descriptors.
 - delete the first S descriptors
 - if the size of the view is still $> c$, remove a set of descriptors chosen at random

Design Choices

<i>Blind:</i>	$H = 0, S = 0$	descriptors to keep are chosen at random
<i>Healer:</i>	$H = c/2, S = 0$	select more fresh descriptors
<i>Swapper:</i>	$H = 0, S = c/2$	delete elements swapped with the neighbours

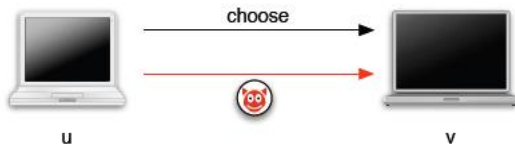
- *Healer*

- deletes links to peers which are no more in the network, with high probability
- goal: delete links connecting to peers which are no more on the overlay, with high probability

- *Swapper*: deletes the peer exchanged with the neighbour, so favouring a continuous refresh of the overlay links

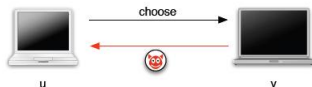
- an high value of S corresponds to a high probability of inserting in the new view the descriptors received from the partner and to discard those sent to the partner
- this strategy favours a high change of the view

The Push Strategy

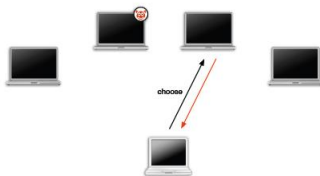


- initially, a few nodes are "infected", the probability to choose a susceptible node is high
- then the probability to find a susceptible node decreases

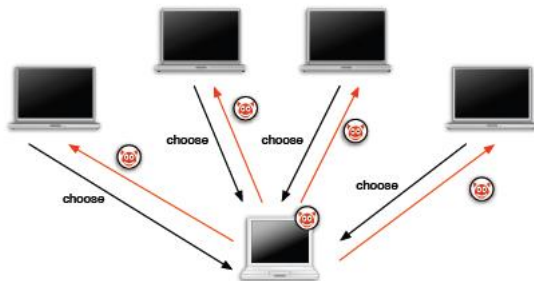
The Pull Strategy



- initially, the probability to be infected is low: a few nodes are infected and must be detected
- there is no guarantee that the diffusion of the information starts at the first cycle

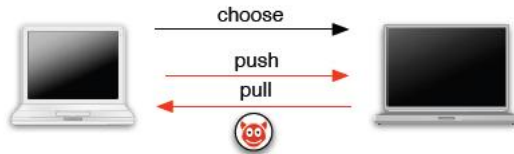


The Pull Strategy



- it is also possible that the whole population is infected in a single cycle

The Push Pull Strategy



- initially the push operations are dominant, because the number of susceptible units is larger of the infective ones
- in the last cycles, the pull operations are dominant, because the probability to meet an infective node is higer
- the probability that a susceptible node contacts an infect node increases at each cycle
- the probability that a infect node contacts a susceptible node decreases at each cycle